نظرية الحوسبة

Theory of Computation

د.ايمن حمارشه

Theory of Computation الحوسبة

كلمة "Computing" أساسًا كانت تستخدم مع ما له علاقة بالعد و الحساب (calculating counting and) أي العلم الذي يتعلق بإجراء الحسابات الرياضية. لكنها لاحقا أصبحت تشير **إلى** عملية الحساب و استخدام الآلات الحاسبة وكذلك العمليات الالكترونية التي تجري ضمن عتاد الحاسب نفسه إضافة **إلى** الأسس النظرية **التي تؤسس لعلوم الحاسب.**

الحوسبة (Computation) : يمكن تعريفها **على** أنها سلسلة الخطوات الوسيطة intermediate steps التي نستخدمها في انجاز خوارزمية مصممة لحل مشكلة **أو مسألة ما** بطريقة حاسوبية .**و**يمكن تعريفها أيضا **على** أنها خوارزمية algorithm نقوم بها لتحويل **مدخلات** input **مسألة ما إلى مخرجات** outputs **أو نتائج** (**أي حل المسألة المطروحة**). وهذا يعني **أن** الحاسوب يقوم بعملية حوسبة computation عندما ينجز برنامجـــــــــــــــــــــا مــــــــــــــــــــا.

وكما هو معروف **في أي** خوارزمية هناك مجموعة من العمليات الحسابية و المنطقية المتسلسلة , **و**نتيجة كل عملية تستخدم كمدخل للعملية التالية , و يقوم البرنامج المعطى الممثل للخوارزمية بترتيب العمليات و تحديد شروط الانتقال من عملية **لأخرى و كذلك** إمكانية العودة **إلى** عملية سابقة **أو الانتقال إلى** عملية لاحقة ليست تالية (**القفز إلى أعلى و إلى أسفل**).

وبناء على ما سبق يمكن تعريف الحوسبة بأنها إيجاد حلول **لمسألة ما ابتداء** من معطيات مطروحة لها باستخدام خوارزمية . ويمكن **أن** يشمل هذا أيضا إيجاد الخوارزميات **المناسبة لحل نمط** معين من المسائل .

أي أن نظرية الحوسبة تبحث في تحليل المسائل ومدخلاتها Inputs إضافة للخوارزميات Algorithms المطروحة لحلها. هذه التعريفات كلها تشكل أساسا لنظرية الحسوبية Computability Theory ونظرية التعقيد الحسابي Computational Complexity Theory .

نظرية **التحسب** Computability Theory: هي أحد فروع المعلوماتية النظرية theoretical computer science **التي تدرس المسائل** القابلة للحل حاسوبيا computationally solvable **باستخدام نماذج مختلفة للحوسبة.**

نظرية الحسوبية تختلف عن التخصصات المشابهة **ك**نظرية التعقيد الحسابي computational complexity theory , حيث **أن** هذه الأخيرة تتعامل مع كيفية حل المسألة حاسوبيا بفعالية , في حين **أن** النظرية الحسوبية تبحث في ما **إذا كانت** هذه المسائل قابلة **أساسا للحل** solvable **باستخدام الحاسوب.**

نظريـة التعقيـد الحسابي Computational Complexity : إحدى أجزاء نظريـة الحوسبة و تتعامل مـع المـوارد المطلوبـة فـي عمليـة الحوسبة . وأكثـر هذه المـوارد شيوعا هـي الـزمن (بمعنـى كـم مـن الخطـوات أو ما يقابلها من الوقت يلزم لحل المسألة) و المكان (بمعنى ما حجم الذاكرة اللازمة لحل المسألة) , ويمكن أن يدخل بالاعتبار موارد أخرى , مثل : كم عدد المعالجات المتوازية اللازمة لإنجاز الحساب باستخدام برمجة متوازية.

في نظرية التحسيب ، تستخدم غالبا الآلات المجردة ضمن التجارب الفكرية المتعلقة بالحسوبية و تحليل تعقيد الخوارزميات (نظرية التعقيد الحسابي) . تعتبر الآلة المجردة النموذجية تتألف من دخل و خرج و مجموعة عمليات مصرح بها تستعمل لتحويل الدخل إلى خرج . أفضل الأمثلة و أكثرها شيوعا هو آلة تورينغ .

لذلك عند إنجاز دراسة منهجية للتحسيب ، يشكل علماء الحاسوب نماذج رياضية مجردة من الحواسيب تدعى نماذج الحوسبة models of computation . توجد عدة أنماط من هذه النماذج قيد الاستعمال ، لكن أهمها وأكثرها شيوعا كما ذكرنا هو آلة تورينغ Turing machine . ويمكن أن نتصور آلة تورينغ على أنها حاسوب منزلي مع سعة ذاكرة محدودة بحيث لا يمكن الوصول إلا إلى قطاعات صغيرة متفرقة من هذه الذاكرة. وتعتبر آلات تورينغ سهلة التصور و التصميم و من الممكن تحليلها و دراستها للبرهنة عن النتائج المتوقعة بالتالي تمثل نموذجا معقولا لعملية الحوسبة .

كما يمكن تعريف آلات مجردة أكثر تعقيدا بمجموعة تعليمات أوسع ، مثل المسجلات و نماذج ذاكرة الحاسوب المختلفة. وأحد أكثر النماذج شيوعا و مشابهة للحاسوب في وضعه الحالي يدعى نموذج RAM ، الذي يسمح بوصول عشوائي لمواقع الذاكرة المفهرسة .

وعندما تكبر فوارق الأداء بين المستويات المختلفة للذاكرة المخبأة (الكاش) cache memory ، تزداد أهمية النماذج الحساسة للكاش مثل نموذج الذاكرة الخارجية external-memory model .

عمليات الحوسبة Computations يتم تصميمها لمعالجة المعلومات المختلفة حيث يمكن أن تكون هذه الحسابات بسيطة كحساب كم يستغرق من الوقت قطع مسافة ما بالسيارة كما يمكن أن تكون معقدة كالحسابات التي تجرى للتنبؤ بالأحوال الجوية ومعرفة حالة الطقس.

وتهدف دراسة الحوسبة إلى الوصول لفهم عميق لخصائص وميزات العمليات الحسابية وهذا الفهم العميق يمكن أن يستخدم في التنبؤ بمدى صعوبة العمليات الحسابية المطلوب إجراؤها وذلك من اجل اختيار الطرق والوسائل المناسبة لتطوير لتسهيل عملية تصميم هذه الطرق.

وتظهر دراسة الحوسبة أن هناك أنواع من المسائل لا يمكن حلها (أي غير قابلة للحل). أما المسائل القابلة للحل فإن بعضها يتطلب عمليا موارد لا يمكن توفيرها (مثلا ملايين السنين من وقت الحساب). ومع أن هذا قد يدعو إلى الإحباط إلا أن محاولة توفير هذه الموارد له ايجابية كبيرة في التحذير من مغبة محاولة حل هكذا نوع من المسائل وهذا لا يتحقق إلا من خلال دراسة الحوسبة التي تساعد في تحديد هذا النوع من المسائل كما تساعد في إيجاد الأدوات المناسبة لتحديد المسائل التي يمكن حلها عمليا وبشكل مناسب بالإضافة إلى إيجاد أدوات تصميم هذه الحلول. كما يمكن من خلال الحوسبة تطوير مصطلحات دقيقة معرفة جيدا للوصول إلى إدراك أو معرفة مسبقة عن هذه المسائل وطرق حلها.

إن القدرة على تمثيل المعلومات يعتبر أمرا حاسما في عملية تبادل ومعالجة هذه المعلومات. لذلك عندما احتاجت المجتمعات البشرية لتبادل المعلومات حتى تتمكن من التواصل فيما بينها فإنهاقامت بإيجاد لغات محكية لتحقيق التواصل المنشود وعندما أصبحت بحاجة إلى مستوى أعلى واعقد من التواصل قامت بتطوير الكتابة writing .

فاللغات المختلفة في شكلها المحكي تعتمد على مجموعات من الأصوات الرئيسية المحدودة كأساس للغة. أما الكلمات فيمكن تعريفها على أنها عبارة عن متواليات محدودة من هذه الأصوات. أما الجمل والعبارات فتنشأ عن متواليات محدودة من الكلمات وفي النهاية فإن الحديث أو الكلام ينشأ من متواليات محدودة من العبارات. واللغة المكتوبة تستخدم مجموعات محدودة من الرموز الأساسية المحدودة.

وبنفس الطريقة تم تطوير أشكال محددة لتمثيل عناصر من مجموعات أخرى. فمثلا الأرقام الطبيعية يمكن تمثيلها بمتواليات محدودة من الأرقام العشرية.والحوسبة مثلها مثل اللغات الطبيعية يتوقع منها أن تتعامل مع المعلومات في أكثر صورها عمومية. وبناءا عليه فإن الحوسبة تتعامل مع الأعداد الصحيحة, الرسوم البيانية, البرامج وأنواع أخرى من الكينونات. وهذا يؤدي إلى استنتاج أن الحوسبة تتعامل فقط مع متواليات من الرموز التي تمثل الأشياء.

<u>الأبجديات (Alphabets) والمتسلسلات (Strings) :</u>

المجموعة المرتبة الغير خالية تسمى أبجدية إذا كانت عناصرها عبارة عن رموز أو حروف لها إشكال تمثيلية تعرف بها. أما التوالي المحدود للرموز المأخوذة من الأبجدية فيسمى متسلسلة.

إذا كانت المتسلسلة تحتوي على الحروف المتتالية **a1,a2,.....,an** فيمكن **أن** نرمز لها بالرمز **a1a2-an** .
أما المتسلسلة التي تحتوي على صفر من الرموز فإنها تسمى متسلسلة خالية **Empty String** ويرمز لها **بالرمز** ε .

مثل : إذا كانت Σ1={a,.....,z} **وكانت** Σ2={1,.....,9} **فإن**:

abb هي متسلسلة تنتمي **إلى** Σ1

123 هي متسلسلة تنتمي **إلى** Σ2

ba12 ليست متسلسلة تنتمي **إلى** Σ1 لأنها تحتوي على رموز غير موجودة **في** Σ1

... 314 ليست متسلسلة لأنها ليست محدودة

المتسلسلة الخالية تنتمي **إلى أي** أبجدية

المجموعة الخالية Ø **لا تعتبر** أبجدية لأنها لا تحتوي على أي عنصر

مجموعة الأعداد الطبيعية ليست أبجدية لأنها ليست محدودة.

اتحاد الأبجديتين Σ1 **و** Σ2 (Σ1 U Σ2) يسمى أبجدية **فقط في حالة إذا أخذت** عناصره ترتيبا معينا.

الأبجدية التي تحتوي على عنصرين فقط تسمى أبجدية ثنائية **Binary Alphabet والمتسلسلة التي تنتمي إلى** هذه الأبجدية تسمى متسلسلة ثنائية, **أما إذا احتوت الأبجدية على عنصر واحد فقط فتسمى** أبجدية أحادية **Unary Alphabet والمتسلسلة التي تنتمي** إليها تسمى متسلسلة أحادية **Unary String** .

طول المتسلسلة (String Length) فهو يساوي عدد الرموز التي تحتوي عليها المتسلسلة **وإذا لم تحتوي المتسلسلة على** أية رموز فتسمى متسلسلة خالية **Empty String** ويرمز لها ε وطولها في هذه الحالة = 0 .
وتلعب المتسلسلة الخالية نفس الدور الذي يلعبه الرقم 0 **في مجموعة الأعداد**.

ويرمز لطول المتسلسلة **X** على سبيل المثال بالرمز | **X** | .

مثل : إذا كانت {0,1} أبجدية ثنائية **و** {1} أبجدية أحادية:**فإن** 11 هي متسلسلة ثنائية تنتمي **إلى الأبجدية** الثنائية وفي نفس الوقت هي متسلسلة أحادية **تنتمي إلى الأبجدية** الأحادية أما طول هذه المتسلسلة فيساوي 2 .

إذا كانت X متسلسلة طولها n فيمكن كتابتها على النحو التالي X=X1X2 .....Xn, حيث Xi **تنتمي إلى** Σ.
وإذا كتبنا هذه الرموز بشكل عكسي فتسمى متسلسلة عكسية Reverse String ويرمز لها بالرمز X **وتكتب** رموزها على النحو **التالي** : X = XnXn-1 ......X1 .

مثال : إذا كانت X = abcd **فإن** X = dcba .

المتسلسلة Z تسمى متسلسلة فرعية Substring **من** X **إذا كانت تتكون من توالي رموز موجودة في** X .

مثال : المتسلسلة cad هي متسلسلة فرعية من المتسلسلة abracada .

إذا كان لدينا المتسلسلة X وطوله m **والمتسلسلة** Y وطولها n **فإن** XY هو تتابع **أو توالي** Concatenation المتسلسلتين ويتم الحصول على XY **من إلحاق** Y بنهاية X **أي:** XY=X1X2....XmY1Y2...Yn . كما يمكن تكوين توالي من المتسلسلة نفسها عدة مرات فمثلا X هي **عبارة عن توالي** X **عدد** k **من المرات.**

مثال : إذا كانت X=01 , Y=100 **فإن** XY=01100

أما توالي المتسلسلة الخالية ε **مع** X فيساوي X **دائما** . **وتوالي** ε مع نفسها يساوي **دائما** ε .

مثال : إذا كانت X=01 **فإن** X = ε , X =01 , X =0101 , X =010101 .

إذا كانت X=011 **فإن المتسلسلات** 011, 101, 110 هي متسلسلات تبادلية Permutations Strings **لـ** X .

مجموعة جميع المتسلسلات التي تنتمي **إلى الأبجدية** Σ يتم تمثيلها بالرمز Σ* . **أما** Σ =Σ* - {ε} .

مثال : إذا كانت X=011 **فإن** 011, 11, 01, 1, 0, ε هي متسلسلات فرعية من X

أما المتسلسلات 01, 0, ε **فتسمى متسلسلات بادئة تامة** Proper Prefixes Strings

والمتسلسلات 11, 1, ε **فتسمى متسلسلات لاحقة تامة** Proper Suffixes Strings .

تنظيم المتسلسلات Ordering of Strings : تعتبر عملية البحث Searching أكثر العمليات التطبيقية العامة التي يتم إجراؤها **على المعلومات** Information وتكمن أهمية هذه العمليات في تسهيل عملية البحث **عن المعلومات** وفي عملية تنظيمها أيضا **بغض النظر عن** الطريقة التي تتم فيها هذه العمليات. **وفي معظم الحالات فإن الوصول إلى** هذه الأهداف يعتمد على وجود علاقات يتم من خلالها تعريف عملية تنظيم مكونات **أو** كينونات المسألة المراد البحث فيها. وفيما يتعلق بالمتسلسلات فإن العلاقة التي يتم تكرارها هو مقارنتها أبجديا كما يحدث في تنظيم الأسماء في دليل الهاتف. ويمكن توضيح ذلك من خلال المثال التالي:

نفرض أن هناك أبجدية هي $\Sigma=\{0,1\}$ **وإذا أخذنا المتسلسلة** 01 **التي تنتمي إلى** هذه الأبجدية فإننا نجدها أبجديا **أصغر من المتسلسلة** 01100 **التي تنتمي** أيضا **إلى** Σ **لأن** 01 يمكن اعتبارها بادئة تامة من 01100 **(أي جزء** منها). من ناحية **أخرى فأن المتسلسلة** 01100 أبجديا **أصغر من** 0111 وذلك لأن المتسلسلتين تشتركان في **أول ثلاثة رموز ولكن الرمز الرابع في** 0111 **اكبر من الرمز الرابع في** 01100.

مثال آخر هو مجموعة متسلسلات تنتمي **إلى** Σ **مرتبة** أبجديا **من الأصغر إلى** الأكبر و لا يزيد طول أي منها عن 3

$$\Sigma^* = \{\varepsilon,0,00,000,001,01,010,011,1,10,100,101,11,110,111\}$$

هذا الترتيب **الأبجدي** يعتبر **أمرا** مريحا للمجموعات المحدودة **أو** المنتهية لأنه يمكن الوصول **إلى** أي من هذه المتسلسلات في هذه المجموعة في نهاية الأمر بغض النظر عن مكان وجودها.

في بعض الحالات يكون الترتيب **الأبجدي** غير مناسب لأن بعض المتسلسلات في المجموعة قد تكون مسبوقة بعدد غير محدود من المتسلسلات **الأخرى. فمثلا في المجموعة الأبجدية المظمة** $\Sigma=\{0,1\}$ **فإن المتسلسلة** 1 **تكون مسبوقة بالمتسلسلات** ....,0,00,000 . وهذا يبرر اللجوء **إلى** ما يسمى التنظيم القانوني Canonical Ordering للمتسلسلات حيث تكون كل متسلسلة مسبوقة بعدد محدود من المتسلسلات **الأخرى**.

وحسب التنظيم القانوني **فإن المتسلسلة** X تعتبر اصغر قانونيا Canonically Smaller **في الأبجدية** Σ^* **من المتسلسلة** Y **إذا تحقق** احد الشرطين التاليين:

1. **إذا كان طول** X **اقل من طول** Y
2. **إذا كانت الاثنتان من نفس الطول ولكن** X **اصغر** أبجديا **من** Y

مثال: في الأبجدية $\Sigma=\{0,1\}$ **والمتسلسلة** X=11 هي اصغر قانونيا من المتسلسلة Y=000 **لأن طول** X **اقل**.

من ناحية أخرى المتسلسلة X=00 اصغر من Y=11 لأنهما من نفس الطول ولكن أبجديا X اقل من Y .

لذلك لو قمنا بترتيب المتسلسلات التي تنتمي إلى الأبجدية في المثال السابق قانونيا وليس أبجديا فإن مجموعة هذه المتسلسلات سوف تكون مرتبة على النحو التالي:

$$\Sigma^* = \{\varepsilon,0,1,00,01,10,11,000,001,010,011,100,101,110,111\}$$

تمثيل المعلومات Representation of Information : من التعريفات والتوضيحات السابقة للأبجديات والمتسلسلات فإن تمثيل المعلومات يمكن النظر إليه على انه مخطط ترتيب الكائنات Objects داخل متسلسلات ولكن حسب قواعد وقوانين معينة.

من ناحية أخرى يمكن القول أن تمثيل أو ترميز Encoding عناصر الأبجدية هو عبارة عن مجموعة ناتجة عن علاقة معينة تسمى دالة تحقق شرطا معينا. وفي حالة كانت الأبجدية مفردة فأن التمثيل يسمى أحاديا وإذا كانت الأبجدية زوجية فأن التمثيل يسمى زوجيا.

مثال: إذا كانت $\Sigma=\{0,1\}$ وكانت الدالة f1 هي تمثيل زوجي لعناصر مجموعة الأعداد الطبيعية كالتالي:

$f_1(0) = \{0, 00, 000, 0000, . . . \}$

$f_1(1) = \{1, 01, 001, 0001, . . . \}$

$f_1(2) = \{10, 010, 0010, 00010, . . . \}$

$f_1(3) = \{11, 011, 0011, 00011, . . . \}$

$f_1(4) = \{100, 0100, 00100, 000100, . . . \}$

وبنفس الطريقة إذا كانت f2 هي أيضا تمثيل لعناصر مجموعة الأعداد الطبيعية على النحو التالي:

$f_2(0) = \{\varepsilon\}$,

$f_2(1) = \{0\}$,

$f_2(2) = \{1\}$,

$f_2(3) = \{00\}$,

$f_2(4) = \{01\}$,

$f_2(5) = \{10\}$,

$f_2(6) = \{11\}$,

$f_2(7) = \{000\}$,

$f_2(8) = \{1000\}$,

$f_2(9) = \{1001\}, \ldots$

وكذلك إذا كانت f3 هي أيضا تمثيل **أحادي للأبجدية** الأحادية $\Sigma=\{1\}$ بحيث يتم تمثيل كل عنصر من مجموعة الأعداد الطبيعية بالعلاقة التالية: $f_3(i) = \{1^i\}$ وفي هذه الحالة فإن :

$f_3(0) = \{\varepsilon\}$,

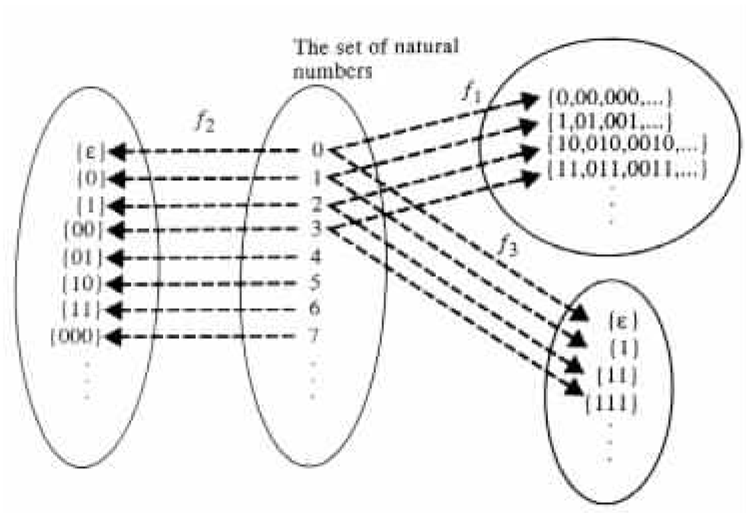
$f_3(1) = \{1\}$,

$f_3(2) = \{11\}$,

$f_3(3) = \{111\}$,

$f_3(4) = \{1111\}, \ldots\ldots$

وبذلك يمكن تمثيل الأعداد في مجموعة الأعداد الطبيعية **بالرسم التالي**:



إن شمولية المتسلسلات تعتبر طريقة مفيدة في تمثيل **Representation** المعلومات طالما وجدت دالة **Function** تقوم بتحويل أو تفسير **Interpretation** المعلومات التي يتم الحصول عليها بواسطة المتسلسلات وهذا التفسير ما هو إلا انعكاس للمخطط الذي توفره عملية التمثيل.

فمثلا المتسلسلة 111 يمكن اعتبارها العدد 111 (مئة وإحدى عشر) حسب ما تمثله متسلسلة عشرية أو الرقم 7 حسب ما تمثله المتسلسلة الثنائية أو الرقم 3 إذا كان ممثلا بمتسلسلة أحادية.

أن الأطراف أو الجهات **Parties** التي تقوم بتبادل جزء من المعلومات هي التي تنجز عمليات التمثيل والتحويل فالتمثيل **Representation** يقوم به المرسل **Sender** أما التحويل **Interpretation** فيقوم به المستقبل **Receiver** وكذلك الأمر بالنسبة للعملية **Process** فليس مهما على الإطلاق إذا كانت هذه الأطراف عناصر بشرية أم برامج **Programs** . لذا فمن وجهة نظر الأطراف المستخدمة فإن اللغة هي فقط عبارة عن مجموعة من المتسلسلات وهذه الأطراف هي التي تضع الحدود لتمثيل وتحويل هذه المتسلسلات.

اللغات Languages :

بشكل عام إذا كانت Σ أبجدية وكانت L مجموعة جزئية **Subset** من Σ^* فأنه يمكن القول أن L هي لغة **Language** تنتمي إلى Σ وكل عنصر في L يسمى جملة **Sentence** أو كلمة **Word** أو متسلسلة **String** .

المجموعات التالية : $\{0,1\}^*$, $\{\varepsilon,10\}$, $\{0,11,001\}$ هي مجموعات جزئية من $\{0,1\}^*$ لذلك فإنها جميعها تعتبر لغات تنتمي إلى الأبجدية $\{0,1\}$.

المجموعة الخالية Ø والمجموعة $\{\varepsilon\}$ هما لغتان تنتميان إلى كل الأبجديات

Ø هي عبارة عن لغة لا تحتوي على أية متسلسلة

{ε} هي عبارة عن لغة تحتوي على متسلسلة خالية

اتحاد Union اللغتين L1,L2 يمثل على النحو التالي L1 U L2 هي عبارة عن اللغة التي تحتوي على جميع المتسلسلات الموجودة في L1 أو في L2 بحيث تحقق {x │x is in L1or X is in L2} .

تقاطع Intersection L1, L2 والذي يمثل L1∩L2 هو عبارة عن اللغة التي تحتوي على جميع المتسلسلات الموجودة في L1 و L2 في نفس الوقت بحيث تحقق {x │x is in L1 and in L2}

متممة Complementation اللغة L التي تنتمي إلى Σ وتمثل كالتالي L هي عبارة عن جميع المتسلسلات الموجودة في Σ وليست موجودة في L بحيث تحقق {x │x is in Σ* but not in L} .

مثال: إذا كانت L1= {ε, 0,1} , L2= {ε, 01, 11} فإن

L1 U L2 = {ε, 0, 1, 01, 11}

L1 ∩ L2 = {ε}

L1 = {00,01, 10, 11, 000, 001, ….}

Ø U L = L

Ø ∩ L = Ø

Ø = Σ*

Σ* = Ø

الفرق Difference بين L1 **و** L2 والذي يمثل (L1-L2) هو عبارة عن جميع المتسلسلات الموجودة في L1 ولكنها ليست في L2 حيث تحقق {x | x is in L1 but not in L2} .

حاصل ضرب Cross Product اللغتين L1 **و** L2 ويمثل ($L_1 \times L_2$) هو عبارة عن المتسلسلة التي تحتوي على جميع الأزواج المرتبة (x,y) بحيث تكون x **من** L1 **وتكون** y **من** L2 **وتحقق** العلاقة التالية:

$\{(x, y) \mid x \text{ is in } L_1 \text{ and } y \text{ is in } L_2 \}$

تركيب Composition L1 **مع** L2 الذي يمثل (L1L2) هو عبارة عن لغة تحقق العلاقة التالية:

$\{ xy \mid x \text{ is in } L_1 \text{ and } y \text{ is in } L_2 \}$

مثل: إذا كانت : L1= {ε,1,01,11} **و** L2={1,01,101} **فإن** :

L1 – L2 = {ε, 11}

L2 – L1 = {101}

مثل: إذا كانت L1={ε,0,1} **وكانت** L2= {01,11} **فإن**:

$L_1 \times L_2 = \{(\varepsilon, 01), (\varepsilon, 11), (0, 01), (0, 11), (1, 01), (1, 11)\}$

$L_1L_2 = \{01, 11, 001, 011, 101, 111\}$

$L - \emptyset = L,$

$\emptyset - L = \emptyset,$

$\emptyset L = \emptyset,$

$\{\varepsilon\}L = L.$

تستخدم L^i لتمثيل التركيب Composition المتكون من العدد i من نسخ اللغة L حيث L^0 تعرف على أنـها {ε} , أما المجموعة $L^0 \cup L^1 \cup L^2 \cup L^3$ ..... فتسمى نـهاية كلين Kleene closure أو ببساطة نـهاية L وتمثل L^* .

أما المجموعة $L^1 \cup L^2 \cup L^3$ .... فتسمى النـهاية الموجبة positive closure للغة L وتمثل $L+$.

إذا كانت $L1= \{\varepsilon,0,1\}$ وكانت $L2=\{01,11\}$ فإن :

$L_1^{\ 2} = \{\varepsilon,0,1,00,01,10,11\}$

$L_2^{\ 3} = \{010101,010111,011111,110101,110111,1111101,111111\}$

جميع هذه العمليات يمكن إجراؤها وبنفس الطريقة على العلاقات في $\sum^* \times \Delta^*$ حيث $\sum , \Delta$ هما أبجديتان.

إذا كانت R1,R2 علاقتان في $\sum^* \times \Delta^*$ فإن:

R1 U R2 فهي تحقق العلاقة $\{ (x, y) \mid (x, y) \text{ is in } R_1 \text{ or in } R_2 \}$

$R1 \cap R2$ تحقق العلاقة $\{ (x, y) \mid (x, y) \text{ is in } R_1 \text{ and in } R_2 \}$

R1R2 تحقق العلاقة $\{ (x_1x_2, y_1y_2) \mid (x_1, y_1) \text{ is in } R_1 \text{ and } (x_2, y_2) \text{ is in } R_2 \}$

مثل : إذا كانت $R1=\{(\varepsilon\ ,0),(10,1) \}$ وكانت $R2= \{(1,\varepsilon),(0,10)\}$ فإن:

$R1 \cup R2 = \{(\varepsilon, 0), (10, 1), (1, \varepsilon), (0, 01)\}$

$R1 \cap R2 = \emptyset$

$R_1R_2 = \{(1, 0), (0, 010), (101, 1), (100, 110)\}$

$R_2R_1 = \{(1, 0), (110, 1), (0, 100), (010, 101)\}$

$\bar{R}$ تحقق العلاقة $\{ (x, y) \mid (x, y) \text{ is in } \Sigma^* \times \Delta^* \text{ but not in } R \}$

R^{-1} تحقق العلاقة $\{ (y, x) \mid (x, y) \text{ is in } R \}$

مثال: إذا كانت العلاقة $R= \{(\varepsilon ,\varepsilon) , (\varepsilon,01)\}$ **فإن :**

$R^{-1} = \{(\varepsilon, \varepsilon), (01, \varepsilon)\}$

$R^0 = \{(\varepsilon, \varepsilon)\}$

$R^2 = \{(\varepsilon, \varepsilon), (\varepsilon, 01), (\varepsilon , 0101)\}$

إن اللغة التي يمكن تعريفها **بواسطة نظام شكلي** **Formal System** ذلك النظام الذي يمتلك عدد محدود من البديهيات **axioms** وكذلك عدد محدود من القواعد والأحكام المستنتجة تسمى لغة شكلية **Formal Language**.

القواعد Grammars :

في كثير من الأحيان يعتبر توصيف اللغة من خلال قواعد محددة أمرا مريحا. وتظهر الفائدة من هذا **–بشكل** رئيسي- **عند استخدام** عدد قليل من القوانين أو الأحكام **Rules** لوصف لغة تحتوي على عدد كبير من الجمل. فعلى سبيل المثال هناك احتمالية أن تتكون الجملة في اللغة الانجليزية من عبارة موضوعية **(مبتدأ) Subject** **تكون متبوعة** بعبارة خبرية **(خبر) Predicate** **أي ان الجملة تتكون من مبتدأ وخبر. وفي نفس الوقت المبتدأ** يمكن أن يتكون من جملة اسمية **Noun Phrase** يتم التعبير عنها بواسطة أحكام كأن يكون المبتدأ اسم **Noun** وهكذا.

بناءاً على ذلك لو أخذنا جملة **"Mary sang a song"** فإنه يمكن وصفها من خلال القواعد أو الأحكام التالية:

< **الجملة** > ← < **مبتدأ** > < **خبر** >

< **المبتدأ** > ← < **اسم** >

< **الخبر** > ← < **فعل** > < **صفة** > < **اسم** >

< **الاسم** > ← < **اسم علم** >

< الاسم > ← < متسلسلة >

< اسم العلم > ← < حرف ابجدي كبير > < متسلسلة >

< المتسلسلة > ← < متسلسلة > < حرف أبجدي >

< المتسلسلة > ← < حرف أبجدي >

< الحرف الأبجدي > ← **a**

< الحرف الأبجدي > ← **z**

< حرف ابجدي كبير > ← **A**

< حرف ابجدي كبير > ← **Z**

< الفعل > ← < يغني >

< أداة التعريف > ← < **a** >

$$\langle \textit{sentence} \rangle \to \langle \textit{subject} \rangle \langle \textit{predicate} \rangle$$
$$\langle \textit{subject} \rangle \to \langle \textit{noun} \rangle$$
$$\langle \textit{predicate} \rangle \to \langle \textit{verb} \rangle \langle \textit{article} \rangle \langle \textit{noun} \rangle$$
$$\langle \textit{noun} \rangle \to \langle \textit{name} \rangle$$
$$\langle \textit{noun} \rangle \to \langle \textit{string} \rangle$$
$$\langle \textit{name} \rangle \to \langle \textit{u_character} \rangle \langle \textit{string} \rangle$$
$$\langle \textit{string} \rangle \to \langle \textit{string} \rangle \langle \textit{character} \rangle$$
$$\langle \textit{string} \rangle \to \langle \textit{character} \rangle$$
$$\langle \textit{character} \rangle \to a$$
$$\vdots$$
$$\langle \textit{character} \rangle \to z$$
$$\langle \textit{u_character} \rangle \to A$$
$$\vdots$$
$$\langle \textit{u_character} \rangle \to Z$$
$$\langle \textit{verb} \rangle \to \textit{sang}$$
$$\langle \textit{article} \rangle \to a$$

إن هذه الأحكام والقواعد تسمح بإنشاء جمل أخرى على غرار جملة “Mary sang a song” وذلك باستخدام اسم آخر إلى جانب Mary او استخدام اسم آخر مكانه. من ناحية أخرى فإن هذه القواعد تشير إلى الجمل التي لا تعتبر من اللغة الإنجليزية مثل جملة “Mary sang a Mary” كما أن هذه القواعد لا تسمح بإنشاء جمل مثل "Mary read a song." ومما سبق فإنه يمكن القول أن مجموعة القواعد والأحكام الواردة أعلاه تكون منظومة قواعد غير مكتملة لوصف اللغة الانجليزية.

ولأغراض البحث فإنه يعتبر كافيا الأخذ بعين الاعتبار فقط القواعد التي تتكون من مجموعات محدودة من القواعد أو الأحكام وهذا النوع من القواعد يسمى قواعد من النوع 0 (Type 0 grammars) أو قواعد بناء التعابير phrase structure grammars أما اللغة الشكلية Formal Language التي تنتجها هذه القواعد فتسمى لغات النوع 0 . وبشكل دقيق يمكن القول أن جميع القواعد من النوع 0 ويرمز لها بالرمز G يمكن تعريفها انها نظام رياضي يتكون من الرباعية $<N,\sum,P,S>$ حيث:

N – أبجدية عناصرها رموز تسمى غير طرفية Nonterminal symbols

∑ - أبجدية عناصرها رموز تسمى طرفية Terminal symbols

P – **عبارة عن** علاقة من تشكيل محدود من $(N \cup \sum^*)$ والتي عناصرها تسمى قواعد انتاج Production Rules. كما أن كل من هذه القواعد (α ,β) **في** P **والتي تمثل على النحو التالي** $\alpha \rightarrow \beta$ يجب أن تمتلك على الأقل رمزا واحدا غير طرفي . وفي كل من هذه القواعد تسمى α ناحية اليد اليسرى left-hand side من القاعدة في حين تسمى β ناحية اليد اليمنى right-hand side **من القاعدة**.

S – هو رمز في N يسمى البداية Start **او الجملة** Sentence .

مثال: إذا كانت <N,∑,P,S> هي Type 0 grammar **وكانت** N={S} , ∑={a,b} , $P=\{S \rightarrow aSb, S \rightarrow \varepsilon\}$ فإن هذه القاعدة تمتلك عنصر واحد nonterminal وهو S , **وعنصران** terminal هما a,b **بالإضافة إلى قاعدتا إنتاج** Production Rules هما $S \rightarrow aSb$ **و** $S \rightarrow \varepsilon$. والقاعدتان من الناحية اليسرى تتكونان فقط من الرمز S . الناحية اليمنى للقاعدة الأولى هي aSb , وللقاعدة الثانية ε .

مثال: <N1,∑1,P1,S> **لا تعتبر قاعدة إذا كانت** N1 مجموعة الأعداد الطبيعية **أو إذا كانت** ∑1 فارغة لأنهما في هذه الحالة لا تعتبران أبجديات.

البرامج Programs :

إن الاعتماد الكبير على معالجة المعلومات قد أدى إلى انتشار واسع للبرامج عند التعامل مع التطبيقات **المختلفة**. فالبرامج تستخدم في البيوت, **أماكن العمل المختلفة**, **الشركات**, المستشفيات, والمدارس وغيرها. ويتم استخدام هذه البرامج في قطاع التعليم, **الألعاب**, عمليات الطباعة والنشر, إدارة الاتصالات الهاتفية, **الفحوصات** الطبية وتشخيص الأمراض, **التنبؤ بالأحوال** الجوية وحالة الطقس, التحكم في الطائرات وغيرها من المجالات.

وقد تم تطوير العديد من لغات البرمجة بغرض تسهيل مهمة كتابة البرامج لمجموعة كبيرة من التطبيقات **المختلفة**. وهذا التنوع في لغات البرمجة يعكس الانطباعات أو التفاسير المختلفة للمعلومات. **وعلى كل حال فمن** جهة القدرة على إجراء عمليات الحوسبة اللازمة فإنه لا فرق كبير بينها. لذلك عند دراسة البرمجة يمكن استخدام **لغات برمجة مختلفة**. أما دراسة البرامج فتفيدنا في عملية اختيار لغة برمجة معينة لاستخدامها في حل مسألة معينة.

عند دراسة البرامج يفضل اختيار لغة برمجة تصلح لحل الكثير من المسائل العامة وفي نفس الوقت أن تكون بسيطة كفاية لتسهيل عملية البحث والدراسة.

اختيار لغة البرمجة Choice of a Programming Language :

يتم تعريف البرنامج على أنه عبارة عن متسلسلة من الأوامر أو التعليمات Instructions في المجال Domain الذي يرمز له D . والمجال D هو عبارة عن مجال من المتغيرات Variables التي يفترض أن تكون مجموعة من العناصر مع وجود عنصر متميز يعطي القيمة الأولية للمتغيرات. ويعتبر كل عنصر من عناصر D قيمة تصلح لتمثيل المتغيرات في البرنامج. أما تسلسل التعليمات فيكون على النحو التالي:

a. **Read instructions of the form** — **قراءة التعليمات في النموذج**

```
read x
```

b. **where x is a variable.** — **حيث x عبارة عن متغير**

c. **Write instructions of the form** — **كتابة التعليمات في النموذج**

```
write x
```

d. **where x is a variable.** — **حيث x عبارة عن متغير**

e. **Deterministic assignment instructions of the form** — **تعليمات مهام إجبارية**

$$y := f(x_1, \ldots, x_m)$$

f. **where $x_1, \ldots, x_m$, and y are variables, and f is a function from D^m to D.**

g. **Conditional if instructions of the form** — **تعليمات الحالة الشرطية if**

$$\texttt{if } Q(x_1, \ldots, x_m) \texttt{ then } I$$

h. **where I is an instruction, $x_1, \ldots, x_m$ are variables, and Q is a predicate from D^m to {false, true}.**

i. **Deterministic looping instructions of the form** — **تعليمات إجبارية للحلقة**

```
Do
     α
until Q(x1, . . . , xm)
```

j. Where α is a nonempty sequence of instructions, $x_1, \ldots, x_m$ are variables, and Q is a predicate from D^m to {false, true}.

k. Conditional accept instructions of the form

```
if eof then accept
```

l. Reject instructions of the form

```
reject
```

m. Nondeterministic assignment instructions of the form

```
x := ?
```

n. where x is a variable.

o. Nondeterministic looping instructions of the form

```
Do
     α1
or
     α2
or
     ⋮
or
     αk
until Q(x1, . . . , xm)
```

p. where k ≥2, each of α1, . . , αk is a nonempty sequence of instructions, x_1, . . . x_m are variables, and Q is a predicate from D^m to {false, true}.

في كل برنامج يفترض أن يكون مجال المتغيرات D تمثيلا لبعض الأبجديات. فعلى سبيل المثال يمكن أن يكون D مجموعة الأعداد الطبيعية,مجموعة الأعداد الصحيحة أو أي مجموعة محدودة من العناصر. أما الدوال Q,f فيفترض أن تكون معطاة ضمن مجموعة الدوال الحسابية المبنية. تبعا لذلك فإن مجال المتغيرات لن يكون مصرح به بشكل واضح إذا كانت قليلة الأهمية.

البرامج التي تكون بدون تعليمات غير قطعية Nondeterministic Instructions تسمى برامج قطعية Deterministic Programs . أما البرامج التي تحتوي على تعليمات غير قطعية فتسمى برامج غير قطعية Nondeterministic Programs .

مثال:

```
read x
y:=0
z:=1
do
     y: = y+1
     z : = z+1
Until z = x
read y
if eof then accept
reject
```

في هذا البرنامج القطعي استخدمت ثلاثة متغيرات أسماؤها x,y,z . كما أن هناك تعليمتان هما الدالة الثابتة f1() = 0 , والدالة الأحادية f2(n)= n+1 التي تقوم بإضافة 1 .

تعليمة الحلقة تستخدم y : = y +1

z : = z + 1

ومن المهم الذكر هنا أنه تم فرض مجموعة الأعداد الطبيعية لمجال المتغيرات مع اعتماد 0 قيمة أولية أو ابتدائية.

أما البرنامج التالي فهو برنامج غير قطعي Nondeterministic Program :

```
do
    read x
or
    y : = ?
    write y
Until y = x
if eof then accept
```

هذا البرنامج يستخدم تعليمتين غير قطعيتين هما تعليمة التصريح على الشكل **"y:=?"** , والأخرى هي تعليمة الحلقة على الشكل **"do-or-until-"** .

أما مدخلات البرنامج فهي عبارة عن تسلسل عناصر من مجال متغيرات البرنامج وكل عنصر من مدخلات البرنامج يسمى قيمة إدخال **Input value** .

نظرية الحوسبة وأقسامها الرئيسية:

عند دراسة نظرية الحوسبة فإنه يجب التركيز على الأقسام الثلاثة الرئيسية التي تكون هذا العلم وهي:

1. **نظرية التشغيل الذاتي Automata**
2. **نظرية التحسيب Computability Theory**
3. **نظرية التعقيد الحسابي Complexity Theory**

هذه الأقسام الرئيسية الثلاث يجمعها سؤال واحد هو :"ما هي القدرات الأساسية للحاسوب وما هي حدود هذه القدرات؟" . وكل واحد من هذه الأقسام يجيب على هذا السؤال من زاوية مختلفة حيث تعتمد الإجابة على طريقة تفسير كل قسم لهذا السؤال.

نظرية التعقيد الحسابي: إن المسائل التي تقوم الحواسيب بحلها تختلف عن بعضها البعض حيث يمكن أن يكون بعضها سهلا والآخر صعبا. فمثلا عملية الفرز والترتيب Sorting Problem تعتبر من العمليات السهلة لذلك فإن ترتيب قائمة من الأعداد ترتيبا تصاعديا تعتبر عملية سهلة بحيث يمكن لأي حاسوب صغير أن يقوم بترتيب ملايين الأعداد وبسرعة كبيرة جدا. ولو قارنا هذه العملية بعملية الجدولة Scheduling Problem فأننا نجد أن الجدولة تعتبر عملية أكثر تعقيدا بسبب وجود قيود على هذه العملية فمثلا عند إعداد جدول المحاضرات الأسبوعي فإنه يجب أن لا يحدث تعارض كوجود صفين في نفس القاعة في نفس الوقت أو وجود أستاذ في صفين مختلفين في نفس الوقت وهكذا. وهنا يبرز سؤال هام وهو "ما الذي يجعل حل بعض المسائل بواسطة الحاسوب صعبة وبعضها الآخر سهلا؟" وهذا هو السؤال المركزي في نظرية التعقيد الحسابي. وتجدر الملاحظة أننا لا نستطيع الاجابة على هذا السؤال رغم الأبحاث والدراسات التي تجري منذ ثلاثة عقود. ومن أهم الإنجازات في نظرية التعقيد الحسابي هو الوصول الى مخطط يمكن من خلاله تصنيف المسائل حسب صعوبة حلها حاسوبيا. بواسطة هذا المخطط يمكن عرض طريقة للدلالة على أن مسألة ما صعبة الحل حاسوبيا حتى ولو لم يكن بالإمكان إثبات ذلك.

وهناك عدة خيارات تظهر عند مواجهة مسألة تعتبر صعبة الحل وهي:

1. **من خلال تحديد وفهم الجانب الصعب من المسألة ومحاولة استبداله مما قد يجعل حل المسألة أكثر سهولة.**
2. **حصر خيارات الحل في حل واحد يكون هو الأمثل وفي بعض الحالات إيجاد حلول تقرب من الحل الصحيح وتجعل الحل صحيح نسبيا.**
3. **بعض المسائل تعتبر صعبة في حالات معينة فقط ولكنها معظم الوقت تعتبر سهلة. وحسب التطبيق Application فإنه قد يكون كافيا أيجاد إجراء Procedure قد يستغرق الوصول إليه زمنا طويلا ولكنه عادة ينفذ بسرعة.**
4. **يمكن اعتماد أنواع بديلة للحوسبة مثل الحوسبة العشوائية Randomized Computation التي يمكنها تسريع بعض المهام Tasks .**

أحد المجالات التطبيقية التي تتأثر مباشرة بنظرية التعقيد الحسابي هو علم التشفير Cryptography وفي أغلب المجالات يتم تفضيل التعامل مع المسائل السهلة على المسائل الصعبة لأن حل هكذا نوع من المسائل يعتبر اسهل وأرخص ثمنا. ويعتبر التشفير شيئا غير عاديا لأنه يتصف بالحاجة إلى حلول حاسوبية صعبة وسبب ذلك أن الرموز السرية أو الأكواد المستخدمة يجب أن تكون صعبة الكسر (أي معرفتها صعبة) بدون الحصول على كلمة السر أو المفتاح. لذلك فإن نظرية التعقيد الحسابي توجه العاملين في مجال التشفير باتجاه المسائل صعبة الحل وتساعد في عملهم لتصميم شفرات جديدة وقوية.

نظرية التحسيب: في النصف الأول من القرن العشرين اكتشف بعض علماء الرياضيات مثل كورت غوديل و آلان تورينغ وغيرهم أن بعض المسائل الأساسية لا يمكن حلها بواسطة الحاسوب. إحدى هذه الظواهر هي مسألة تحديد فيما إذا كانت عبارة رياضية ما صائبة أم خاطئة لأن حل هذه المسألة بواسطة الحاسوب قد يبدو شيئا ممكنا ولكن في الواقع لا توجد أية خوارزمية تستطيع إنجاز هذه المهمة. ونظرا لأهمية هذه النتيجة فقد تم تطوير بعض الأفكار التي تهتم بالنماذج النظرية للحاسوب Theoretical Models of Computers والتي سوف تساعد في إيجاد تراكيب لحواسيب حقيقية.

إن نظرية التحسيب ونظرية التعقيد الحسابي قريبتان جدا من بعضهما ففي نظرية التعقيد الحسابي الموضوع الأساسي هو تصنيف المسائل وفرزها إلى صعبة وسهلة في حين نظرية التحسيب تقوم بتصنيف المسائل إلى مسائل قابلة للحل أو غير قابلة للحل.

نظرية التشغيل الذاتي: تتعامل هذه النظرية مع تعريفات وخصائص النماذج الرياضية. وهذه النماذج تلعب دورا مهما في الكثير من المجالات التطبيقية لعلوم الحاسوب. أحد هذه النماذج يسمى "التشغيل الذاتي المحدود" Finite Automaton والذي يستخدم في معالجة النصوص وفي المترجمات وكذلك في تصميم معدات الحاسوب. نموذج آخر يسمى "قاعدة السياق الحر" Context-Free Grammar والذي يستخدم في لغات البرمجة والذكاء الاصطناعي. وتعتبر دراسة نظرية التشغيل الذاتي بداية مناسبة لدراسة نظرية الحوسبة وذلك لأن نظرية التحسيب ونظرية التعقيد الحسابي تتطلب تعريفات دقيقة للحوسبة أما نظرية التشغيل الذاتي فتسمح بالتطبيق من خلال تعريفات شكلية للحوسبة حيث أنها تقوم بتقديم مبادئ لها علاقة بالجوانب غير النظرية لعلوم الحاسوب.

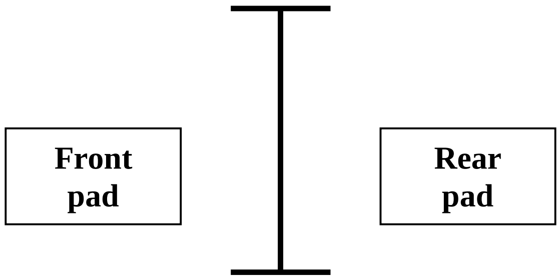
تبدأ نظرية الحوسبة بالسؤال "ما هو الحاسوب؟" قد يبدو هذا سؤالا ساذجا لكن في واقع الأمر فإن الحاسوب الحقيقي معقد جدا لدرجة لا تسمح بتطبيق النظريات الرياضية عليها مباشرة وبدلا من ذلك فإننا نستخدم حاسوبا مثاليا يسمى النموذج الحاسوبي Computational Model .

وكما هو الحال بالنسبة لأي نموذج علمي فإن النماذج الحاسوبية قد تكون دقيقة في بعض الأحيان وغير دقيقة في بعضها الآخر. وبالتالي فإننا سوف نستخدم نماذج مختلفة وذلك حسب الجوانب التي يتم التركيز عليها. وسوف نبدأ بأبسط نموذج والذي يسمى نموذج التشغيل الذاتي المحدود Finite State Machine أو Finite Automaton .

نماذج التشغيل الذاتي المحدودة Finite Automata :

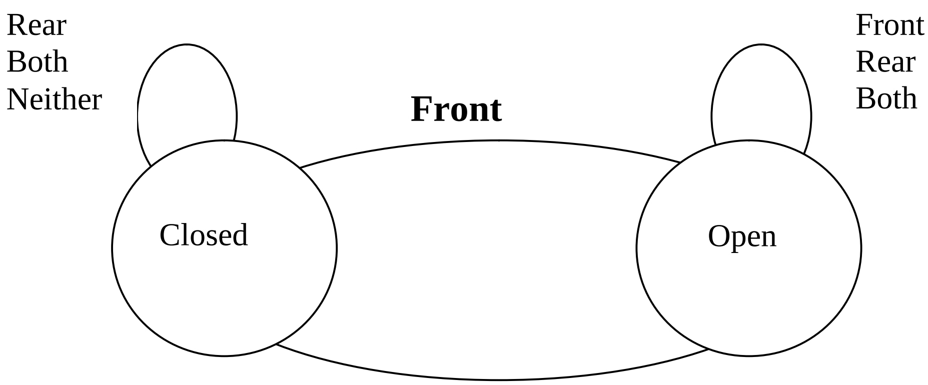
تعتبر هذه النماذج مناسبة للحواسيب التي تمتلك ذاكرة محدودة جدا. وهذا لا يقلل من الأشياء التي يمكن لهكذا حواسيب فعلها. ونحن في واقع الأمر نتعامل مع هكذا حواسيب طول الوقت حيث أنها تقع في القلب من مختلف

الآلات الكهروميكانيكية مثل الأبواب الآلية التي نراها عادة في مداخل المطارات والمحلات التجارية وغيرها **والتي تحتوي على متحكم** Controller يقوم بالتحكم في **فتح الباب** عند اقتراب جسم مادي لمسافة معينة.



Automatic door

في هذه الأنواع من الأبواب تكون هناك منطقة **أمام الباب** (Front Pad) **في حالة تواجد أي جسم مادي** في هذه المنطقة فإن هناك مجسات تشعر بهذا الجسم وتنقل **الإشارة إلى متحكم** Controller الذي يقوم بتحليل وتفسير هذه **الإشارة** ثم يصدر أمرا **إلى** الأجزاء الميكانيكية لتقوم بفتح الباب وهي الحالة Open. **كما أن** هناك **منطقة خلف الباب** (Rear Pad) **إذا** تواجد فيها جسم مادي يقوم المتحكم بإصدار إشارات تبقي الباب مفتوحا بالقدر الكافي من الزمن لكي يعبر هذا الجسم من خلال الباب **وإذا كانت حالة الباب** Open **وتواجد جسم خلف الباب فإن حالة الباب تبقى** Open وذلك حتى لا يتعرض هذا الجسم للأذى **إذا تم إغلاق الباب**. أي أن مهمة المتحكم هو تغيير حالة الباب من Open **إلى** Closed **والعكس**.



الشكل أعلاه هو عبارة عن مخطط حالة متحكم الباب الآلي. ومن هذا المخطط **نلاحظ** أن هناك **حالتان** للمتحكم هي Open **و** Closed **وكذلك أربعة ا**حتمالات للقيم التي يمكن إدخالها إلى هذا النظام. **وبناءا على** هذه المدخلات يقوم المتحكم بتغيير حالته **من** Open **إلى** Close **والعكس**. **و**هذه المدخلات هي:

- **أمام** (Front) وتعني أن هناك جسما ماديا أمام الباب
- **خلف** (Rear) وتعني أن هناك جسما ماديا خلف الباب
- **الاثنان** (Both) **وتعني** أن هناك جسمان أحدهما أمام الباب والثاني خلف الباب
- **لا أحد** (Neither) **وتعني أن لا أجسام أمام أو خلف الباب**

ويوضح هذا المخطط أنه إذا كانت الحالة الابتدائية Closed وتم إدخال إشارة Neither أو إشارة Rear فإن الباب سيبقى في حالة Closed وكذلك الحال إذا كانت الإشارة Both فلن يتم فتح الباب وتبقى الحالة Closed وذلك حتى لا يتعرض للأذى الشخص الموجود خلف الباب إذا تم فتحه. ولكن في حالة كانت الإشارة المدخلة Front فعندها فقط تتغير الحالة إلى Open. أما إذا كانت الحالة الابتدائية Open فإن استقبال الإشارات Front, Rear و Both سيبقي الباب في حالة Open وفقط في حالة واحدة هي استقبال الإشارة Neither يتم الإغلاق.

مثال: إذا كانت حالة المتحكم الابتدائية Closed وتم استقبال الإشارات التالية بالترتيب التالي:

Front → Rear → Neither → Front → Both → Neither → Rear → Neither

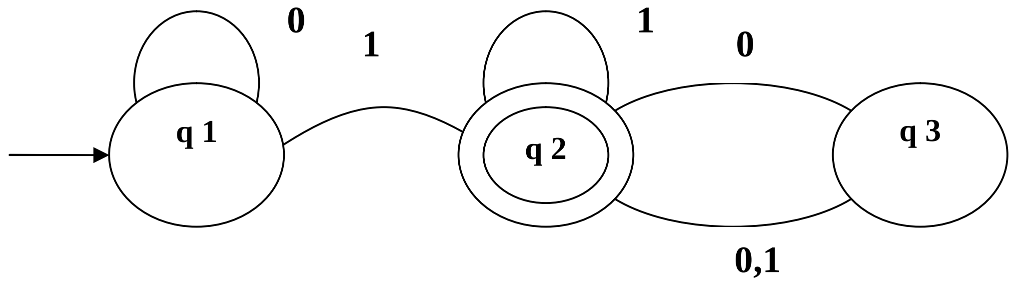
فإن المتحكم سوف ينتقل إلى الحالات التالية على النحو التالي:

Open → Open → Closed → Open → Open → Closed → Closed → Closed

إن دراسة متحكم الباب الآلي كنموذج للتشغيل الذاتي المحدود تعتبر مفيدة وذلك لأنها تقدم طريقة واضحة لتمثيل هذا النوع من النظم كما هو واضح في المثال السابق. فالمتحكم هو عبارة عن حاسوب يمتلك بت واحد من الذاكرة يمكنه تسجيل الحالة التي يكون عليها المتحكم هل هي Close أم Open . وهناك بعض الأجهزة التي تمتلك ذاكرة محدودة ولكنها أكبر قليلا من هذه. فمثلا حالة متحكم المصعد الآلي Elevator يمثلها رقم الطابق الذي يوجد عليه المصعد في لحظة ما. أما المدخلات فيمكن أن تكون الإشارات التي يتم إرسالها بالضغط على أزرار موجودة على لوحات للتحكم في العمليات المطلوبة من المصعد القيام بها. وهذا النوع من المتحكمات يوجد في الكثير من الأجهزة المستخدمة في البيوت مثل الغسالات وأجهزة التبريد والتكييف وكذلك توجد في الآلات الحاسبة والساعات الإلكترونية وغيرها.

لوصف النظريات الرياضية للتشغيل الذاتي المحدود فإنه يجب فعل ذلك بطريقة مجردة أي دون ربط ذلك بأية تطبيقات محددة.

الشكل التالي يصور نموذجا للتشغيل الذاتي يسمى M1 :



نلاحظ من هذا الشكل الذي يسمى مخطط حالة M1 أن لهذا النموذج ثلاث حالات تمثل بالرموز q3,q2,q1 حيث:

- q1 تسمى الحالة الابتدائية Start State
- q2 تسمى حالة القبول Accept State

- q3 تسمى حالة الرفض Reject State

أما الأسهم في المخطط فهي تمثل عملية الانتقال من حالة إلى أخرى وتسمى مقاطع التحول Transitions .

لو فرضنا أن هذا النموذج تلقى مدخلات على شكل المتسلسلة 1101 فإن هذه المتسلسلة يتم معالجتها للحصول على مخرجات تكون إما قبول أو رفض. وسوف نعتبر هذا النوع من المخرجات هو من نوع yes/no بمعنى أن لها حالتان فقط.

تبدأ معالجة المدخلات في هذا النظام في الحالة الابتدائية حيث يتم استقبال عناصر المتسلسلة واحدا تلو الآخر من اليسار إلى اليمين. بعد قراءة كل عنصر ينتقل M1 من حالة إلى أخرى وعند قراءة آخر عنصر يقوم M1 بإنتاج المخرجات. وهذه المخرجات تكون Accept إذا كان M1 في حالة Accept عند قراءة آخر عنصر من المدخلات وتكون Reject إذا كانت حالة M1 هي Reject عند قراءة آخر عنصر من المدخلات.

فمثلا عند إدخال المتسلسلة 1101 إلى الآلة M1 التي يمثل المخطط أعلاه طريقة عملها فإن العمليات تتم على النحو التالي :

1. ابدأ في الحالة q1
2. اقرأ 1 , ويتم الانتقال من q1 إلى q2
3. اقرأ 1 , ويتم الانتقال من q2 إلى q2
4. اقرأ 0 , ويتم الانتقال من q2 إلى q3
5. اقرأ 1 , ويتم الانتقال من q3 إلى q2
6. اقبل , لأن M1 في حالة القبول q2 عند إنهاء الإدخال.

وهذا يعني أن المخرجات سوف تكون حالة Accept .

وعند تجربة عدد كبير من المتسلسلات على هذه النظام يمكن استنتاج التالي:

- جميع المتسلسلات التي تنتهي بالعنصر 1 مثل 1, 01 , 111101 , 0101010101 مخرجاتها accept
- جميع المتسلسلات التي تنتهي بعدد زوجي من الأصفار بعد 1 مثل 100 , 1000000 مخرجاتها accept
- جميع المتسلسلات التي تنتهي بعدد فردي من الأصفار مثل 01000 مخرجاتها reject

التعريف الشكلي Formal Definition لنماذج التشغيل الذاتي المحدودة:

في الأمثلة السابقة قمنا باستخدام مخططات الحالة State Diagrams وذلك لتعريف Finite Automata وذلك لأن هذه المخططات تساعد كثيرا في فهم واستيعاب هذه النظم ومع ذلك فإننا بحاجة إلى استخدام التعريف الرياضي أو ما يسمى التعريف الشكلي Formal Definition وذلك لسببين رئيسيين هما:

1. التعريف الشكلي يكون دقيقا حيث يقوم بإزالة أي غموض أو عدم دقة بالنسبة لما يسمح به في هذه النماذج.
2. التعريف الشكلي يكون على شكل مجموعة رموز مكتوبة وهذا يساعد في التفكير وكذلك في التعبير عن الأفكار بشكل واضح.

يتكون Finite Automaton من مجموعة أجزاء. حيث أن له مجموعة من الحالات ومجموعة قواعد للانتقال من حالة إلى أخرى حسب عنصر الإدخال كما أن له أبجدية إدخال تظهر أية عناصر يسمح بها كمدخلات. بالإضافة إلى ذلك فإنه يمتلك حالة ابتدائية وكذلك مجموعة من حالات القبول. بالاعتماد على ما سبق فإن التعريف الشكلي يقول أن Finite Automaton هو عبارة عن قائمة من هذه المكونات الخمسة.

التعريف الشكلي يستخدم ما يسمى دالة الانتقال Transition Function والتي يتم تمثيلها بالرمز δ وذلك لتعريف قواعد الانتقال. فمثلا إذا كان المخطط يظهر عملية الانتقال من الحالة x إلى الحالة y بواسطة سهم وذلك عند قراءة عنصر الإدخال 1 فإن هذا يعني أنه إذا كان النموذج في حالة x فإنه عند قراءة عنصر الإدخال 1 سوف ينتقل إلى الحالة y . نفس الشيء يمكن التعبير عنه باستخدام دالة الانتقال كالتالي $\delta(x,1) = y$.

التعبير بهذا الشكل هو عبارة عن نوع من الاختصار الرياضي وعند وضع كل المكونات مع بعضها البعض فإننا بذلك نصل إلى التعريف الشكلي لنموذج التشغيل الذاتي المحدود.

يتكون Finite Automaton من الخماسية التالية : $(Q,\Sigma , \delta,q0,F)$ حيث:

1. Q : هي عبارة عن مجموعة محدودة تسمى الحالات States .
2. ∑ : هي عبارة عن مجموعة محدودة تسمى أبجدية Alphabet .
3. δ : هي دالة الانتقال Transition Function حيث $\delta: Q \times \Sigma \longrightarrow Q$
4. q0 : هي الحالة الابتدائية Start State حيث q0 Q
5. F : هي مجموعة حالات القبول Accept States حيث F Q

وهذا يعني أنه يمكن استخدام رموز التعريف الشكلي لأن هذا يعطي وصفا دقيقا لنموذج التشغيل الذاتي المحدود. فمثلا بالنسبة للنموذج M1 الوارد ذكره سابقا فإنه باستخدام الرموز يمكن وصف هذا النموذج على النحو التالي:

$M1 = (Q,\Sigma , \delta,q1,F)$

$\Sigma = \{q1, q2, q3\}$

δ ويتم وصفها على النحو التالي :

	0	1
q 1	q 1	q 2
q 2	q 3	q 2
q 3	q 2	q 2

q 1 : Start State (الحالة الابتدائية)

$F = \{q2\}$

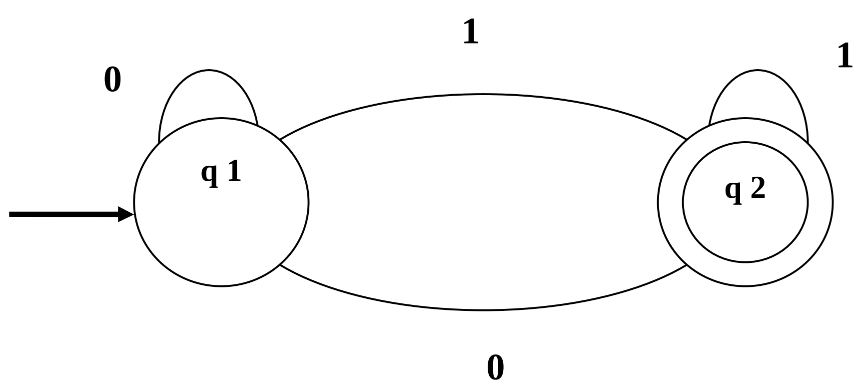
وإذا اعتبرنا A هي مجموعة جميع المتسلسلات التي تقبلها الآلة M فإنه يمكن القول أن A هي لغة الآلة M ويتم كتابة **ذلك على النحو التالي:** L(M) = A كما يمكن القول أن الآلة M يمكنها تمييز أو إدراك Recognize **اللغة** A أو بتعبير آخر أن M **تقبل** A . **وبما أن المصطلح قبول** Accept يحمل معاني كثيرة مختلفة فإنه يفضل استخدام المصطلح Recognize وذلك تفاديا لأي إرباك أو تشويش في فهم المعنى الصحيح.

إن الآلة يمكنها قبول عدة متسلسلات ولكنها دائما تدرك لغة واحدة فقط. وفي حالة كانت الآلة لا تقبل أية متسلسلة فإنها مع ذلك لا تزال تدرك لغة واحدة فقط تسمى اللغة الخالية $\emptyset$. ولهذه الآلة تعرف اللغة على الشكل **التالي:**

A={w/w contains at least one 1 and an even numbers of 0s follow the last 1}

وعندها نقول أن الآلة تدرك اللغة A ويكتب ذلك كالتالي L(M1) = A .

مثال: المخطط التالي يمثل **Finite Automaton** اسمه **M2** :



التعريف الشكلي لهذا النظام سيكون على النحو التالي:

$$M2 = (\{q1, q2\}, \{0,1\}, \delta, q1, \{q2\})$$

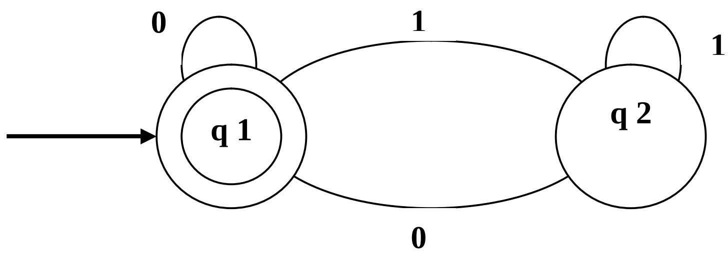
أما دالة الانتقال δ **فتمثل على النحو التالي:**

	0	1
q 1	q 1	q 2
q 2	q 1	q 2

ويجب التأكيد هنا أن مخطط الحالة **State Diagram** والتعريف الشكلي **Formal Definition** يحتويان على نفس المعلومات ولكن بأشكال مختلفة بحيث يمكن الانتقال من احدهما إلى الآخر إذا كان ذلك ضروريا.

إن أفضل طريقة للبدء في فهم أية آلة هي تجربتها من خلال عينة من متسلسلات الإدخال بحيث نرى من خلالها كيف تعمل الآلة وكذلك حتى تصبح طريقة عملها واضحة وجلية. على سبيل المثال إذا أخذنا المتسلسلة 1101 كعينة إدخال فإن الآلة M2 سوف تبدأ عملها في الحالة الابتدائية q1 **ثم تنتقل إلى الحالة** q2 **بعد قراءة** أول عنصر في المتسلسلة وهو 1 **ثم تنتقل إلى الحالات** q2, q1, q1 **على التوالي بعد قراءة باقي عناصر المتسلسلة**. هذه المتسلسلة تم قبولها لأن q2 هي حالة القبول والتي توجد فيها الآلة بعد قراءة آخر عنصر. **ولكن لو أخذنا المتسلسلة** 110 كعينة إدخال فإن تنفيذها سوف يجعل الآلة في حالة q1 وهي حالة الرفض بعد قراءة آخر عنصر فيها أي أن المتسلسلة قد تم رفضها. وبعد تجربة أكثر من عينة إدخال نصل إلى نتيجة مفادها أن M2 تقبل جميع المتسلسلات التي تنتهي بالعنصر 1 ويمكن صياغة هذا كالتالي: L(M2) = {w/w ends in a 1}
أي أن لغة هذه الآلة جميع المتسلسلات التي تنتهي بالرمز 1 .

مثال : المخطط التالي يمثل Finite Automaton **ولتكن الآلة** M3 :



هذه الآلة مشابهة جدا للآلة M2 **في المثال السابق** سوى أن حالة القبول هنا هي q1 وحالة الرفض هي q2 .
التعريف الشكلي لهذه الآلة هو :

$$M3 = (\{q1, q2\}, \{0,1\}, \delta, q1, \{q1\})$$

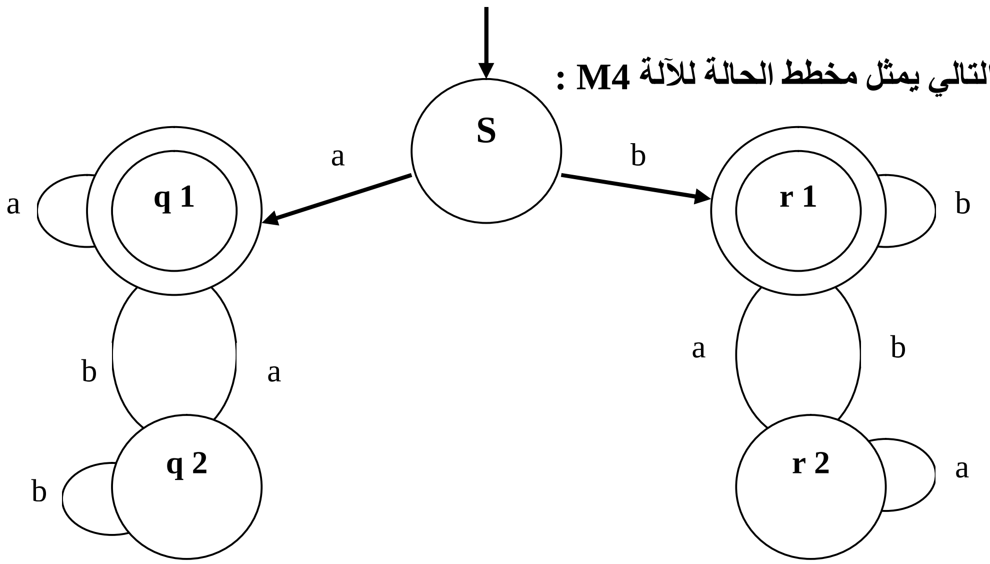
أما دالة الانتقال δ فيتم تمثيلها كالتالي :

	0	1
q 1	q 1	q 2
q 2	q 1	q 2

كما نعلم أن الآلة تقبل المتسلسلة التي تجعلها في حالة قبول بعد قراءة آخر رمز من رموز هذه المتسلسلة. ونلاحظ هنا أن الحالة الابتدائية لهذه الآلة هي حالة القبول وهذا النوع من الآلات يقبل المتسلسلة الخالية ε لأنه بعد قراءة هذه المتسلسلة تبقى الآلة في نفس الوضع أي **وضع القبول**. بالإضافة إلى هذا فإن الآلة تقبل أية متسلسلة تنتهي بالرمز 0 . لذلك فإن لغة هذه الآلة يتم التعبير عنه على النحو التالي:

L(M3) = {w/w is the empty string ε or ends in a 0}

مثال: الشكل التالي يمثل مخطط الحالة للآلة M4 :



يظهر من هذا المخطط أن الآلة تمتلك خمسة حالات هي s, r1, r2, q1, q2 وكذلك حالتا قبول هما q1, r1 أما الحالة الابتدائية فهي s. وعلى كل حال فإن التعريف الشكلي لهذه الآلة هو:

M4 = ({s, r1, r2, q1, q2}, {a,b}, δ , s, {q1, r1}

أما دالة الانتقال فنصفها كالتالي:

	a	b
s	q 1	r 1
r 1	r 2	r 1
r 2	r 2	r 1
q 1	q 1	q 2

q 2	q 1	q 2

هذا النوع من الآلات يبدأ في **الحالة** s **وبعد قراءة أول رمز في المدخلات فإن الآلة ستنتقل إما إلى** اليمين أي **إلى الحالة** r1 أو إلى اليسار إلى q1 وفي الحالتين فإنها لن تعود إلى الحالة الابتدائية أبدا.

ويعتمد اتجاه عمل الآلة على أول رمز في متسلسلة الإدخال **فإذا كان** a ستذهب إلى الحالة q1 وهي **حالة القبول** وستعود أيضا إلى حالة القبول إذا كان آخر رمز في المتسلسلة هو a أيضا أي أن أية متسلسلة تبدأ وتنتهي بالرمز a سيتم **قبولها**. **أما إذا كان أول رمز** b فسيكون الاتجاه إلى الحالة r1 **والمتسلسلة التي تبدأ** وتنتهي بالرمز b سوف تكون متسلسلة قبول أيضا. وفي هذه الحالة نقول أن المتسلسلة التي تبدأ وتنتهي بنفس الرمز هي متسلسلة قبول. على سبيل المثال المتسلسلات التالية bab, aba, bb, aa, b, a جميعها متسلسلات قبول على العكس من المتسلسلات التالية bbba, ba, ab فجميعها لا يتم قبولها. أما لغة هذه الآلة فتعريفها هو:

L(M4) = {w/w is the string that begins and ends with the same symbol }

مثال: مثالنا التالي هو عبارة عن متحكم مصعد آلي elevator controller يقوم بخدمة طابقين فقط. مدخلات هذا النظام هي عبارة عن استدعاءات calls يتم ادخالها إلى النظام **من داخل المصعد أو من الطوابق** وهذه المدخلات هي:

0 – no calls (لا يتم استدعاء المصعد)

1 – call to floor 1 (**استدعاء إلى الطابق الأول**)

2 – call to floor 2 (**استدعاء إلى الطابق الثاني**)

هذا المصعد يمكنه الصعود إلى أعلى , الهبوط إلى أسفل وكذلك الانتظار على أحد الطوابق. **في حالة كان المصعد** على طابق ما فإنه يمكن أن يكون في حالة انتظار استدعاء أو أن يكون على وشك الانتقال إلى طابق **آخر**. هذا يقودنا إلى أن هناك ست حالات يمكن للمصعد أن يكون فيها وهي:

W1 – waiting on first floor (**انتظار على الطابق الأول**)

U1 – about to go up (**على وشك الصعود**)

UP – going up (**حالة الصعود**)

DN – going down (حالة الهبوط)

W2 – waiting on second floor (**انتظار على الطابق الثاني**)

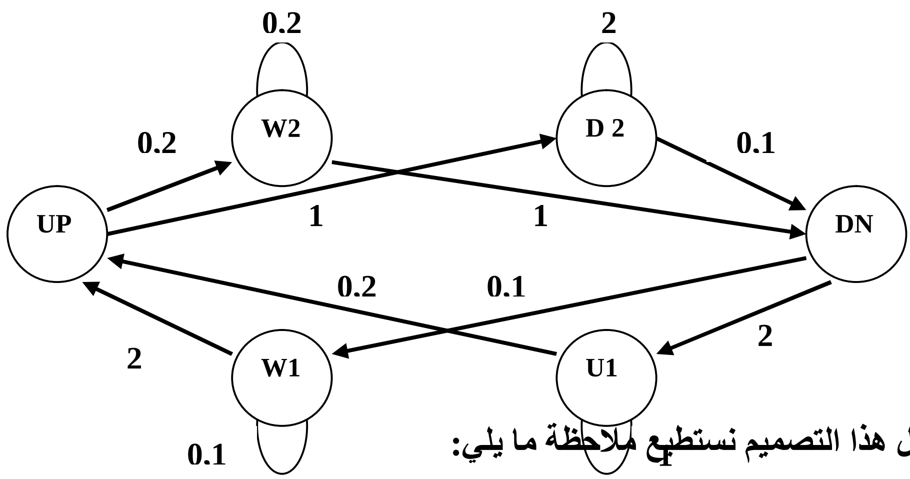
D2 – about to go down (على وشك الهبوط)

وحسب الإشارات المرسلة للمصعد وردة فعل المصعد عليها فإنه يمكن تكوين جدول الحالات التالي:

State	Input		
	0	1	2

W1	W1	W1	UP
U1	UP	U1	UP
UP	W2	D2	W2
DN	W1	W1	U1
W2	W2	DN	W2
D2	DN	DN	D2

بناءً على هذا الجدول يمكن تصميم مخطط الحالة لمتحكم المصعد الآلي والذي سيكون على النحو التالي:



من خلال هذا التصميم نستطيع ملاحظة ما يلي:

- **إذا كان المصعد في حالة على وشك التحرك سواءً إلى أعلى أو إلى أسفل (الحالات** D2, U1) **وتم استدعاؤه إلى** نفس الطابق الموجود عليه فعلا فإنه سيظل على هذا الطابق وسيبقى في نفس حالته.
- إذا كان المصعد في حالة صعود أو هبوط **(الحالات** DN, UP) **وتم استدعاؤه** إلى الجهة المعاكسة لاتجاه حركته فإنه سيخزن الاستدعاء بالذهاب إما إلى U1 **أو** D2 . **(مثلا إذا كان في حالة صعود من الطابق الأول إلى الثاني** UP وتم استدعاؤه خلال ذلك إلى الطابق الأول فإنه سوف ينتقل إلى حالة على وشك الهبوط D2 عند وصوله الطابق الثاني).
- لتسهيل عملية فهم عمل النظام لم يتضمن هذا التصميم حالات أخرى يفترض وجودها في النظام مثل عملية فتح وإغلاق أبواب المصعد, انقطاع التيار الكهربائي, زيادة الوزن عن القدر المحدد وكذلك تعطل **المصعد.**
- حالات القبول والرفض ليست موضوع نقاش في النظام لذلك لم يتم التطرق إليها.
- يمكن تصميم هذا النظام والأخذ بعين الاعتبار الحالات التي لم يتم التطرق إليها وهذا يجعل التصميم أعقد وأكثر صعوبة وخاصة إذا تصورنا المصعد الآلي الذي يخدم 100**طابق مثلا.**

تصميم نماذج التشغيل الذاتي المحدودة Designing Finite Automata :

إن التصميم بحد ذاته يعتبر عملية إبداعية بغض النظر عن الشيء الذي نقوم بتصميمه وذلك لأنه من غير الممكن إيجاز أو اختصار عملية التصميم في وصفة بسيطة أو صيغة محددة. ومع ذلك يمكن إيجاد بعض المقاربات والتفصيلات التي تساعد في عملية تصميم الأنواع المختلفة من نماذج التشغيل الذاتية. فمثلا يمكن للمصمم أن يتخيل كيف سيكون سير العمليات المطلوب من الآلة إنجازها أي أن يضع نفسه مكان هذه الآلة وكيف

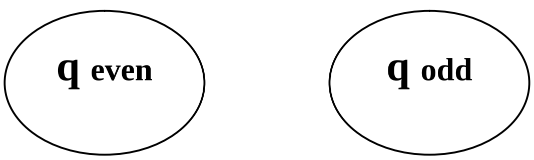
ستعمل هذه الآلة. هذا الافتراض هو حيلة نفسية تساعد في تسخير كامل القدرات العقلية للمصمم في عملية التصميم.

لو فرضنا أننا نريد تصميم آلة ذاتية الحركة Finite Automaton بحيث تكون هذه الآلة قادرة على إدراك لغة ما وهذا يعني أنه عندما تبدأ الآلة باستقبال متسلسلة الإدخال فإن عليها أن تقرر إذا ما كانت هذه المتسلسلة جزءاً من اللغة التي يفترض أن تدركها الآلة. رموز المتسلسلة يتم قراءتها - كما هو معروف – **واحدا تلو الآخر** وبعد قراءة كل رمز يجب أن تكون الآلة قادرة على الإجابة عن السؤال "هل المتسلسلة جزء من هذه اللغة؟" والسبب هو أن الآلة لا تعرف متى ستنتهي هذه المتسلسلة لذلك يجب أن تكون الآلة جاهزة دائما وفي أية لحظة للإجابة على هذا السؤال. للإجابة على هذا السؤال يجب أولا تحديد ما هي الأشياء التي يجب على ذاكرة الآلة حفظها أو تخزينها عن المتسلسلة. وبما أن هذا النوع من الآلات يمتلك ذاكرة محدودة جدا لأنها تمتلك عدد محدود **من الحالات** States وهذا يعني أن الآلة لا تستطيع تذكر أو تخزين كمية كبيرة من البيانات. **ولو فرضنا أن** مدخلات هذه الآلة هي متسلسلة طويلة يبلغ طولها ملايين الكيلومترات فإنه من المستحيل على هذا النوع من **الآلات** حفظ أو تذكر جميع رموز المتسلسلة. **ولحسن الحظ** فإنه بالنسبة للعديد من اللغات ليست هناك ضرورة لحفظ المدخلات كلها وإنما يكفي فقط حفظ بعض المعلومات المهمة. أما تحديد ما هي المعلومات التي تعتبر مهمة فيعتمد على خصوصية اللغة ذاتها بمعنى أن لكل لغة معلومات مهمة قد لا تكون مهمة في لغة أخرى.

على سبيل المثال لو أردنا تصميم Finite Automata **ولتكن** E1 بحيث تستطيع هذه الآلة إدراك لغة تنتمي رموزها إلى الأبجدية {0,1} وتتكون هذه اللغة من جميع المتسلسلات التي تحتوي على عدد فردي من الآحاد. فإنه بداية يجب أن تبدأ هذه الآلة باستقبال متسلسلة الإدخال **المكونة من الأصفار والآحاد رمزا تلو الآخر**. والسؤال المهم هنا "هل يجب على هذه الآلة تذكر (**حفظ**) جميع رموز متسلسلة الإدخال؟" **حتى تقرر** هل عدد **الآحاد** فيها فردي. والإجابة هي لا. وذلك لأنه ببساطة يكفي فقط حفظ معلومة مهمة واحدة هي "هل عدد الآحاد التي تمت قراءتها حتى الآن فردي أم زوجي؟" ومع قراءة كل واحد جديد سوف تتغير القيمة المخزنة.

عندما تبدأ الآلة في قراءة المدخلات فإن المعلومات المهمة التي يجب حفظها **عن** المتسلسلة هي عبارة عن قائمة من احتمالين هما 1. **العدد زوجي** 2. **العدد فردي**

تبعا لذلك فإنه سيتم اختيار حالات الآلة E1 **حسب المخطط التالي**:

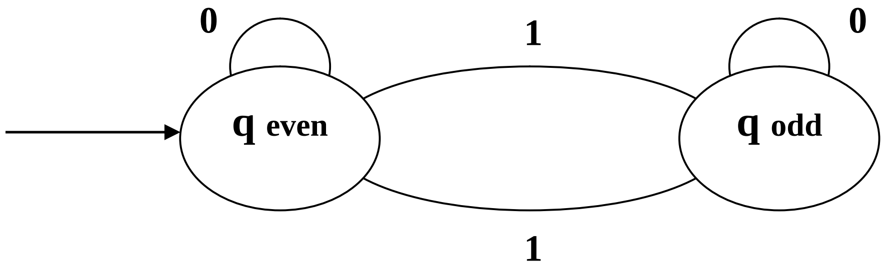


حيث : q even تمثل الحالة التي يكون فيها عدد الحالات التي تمت قراءتها زوجي

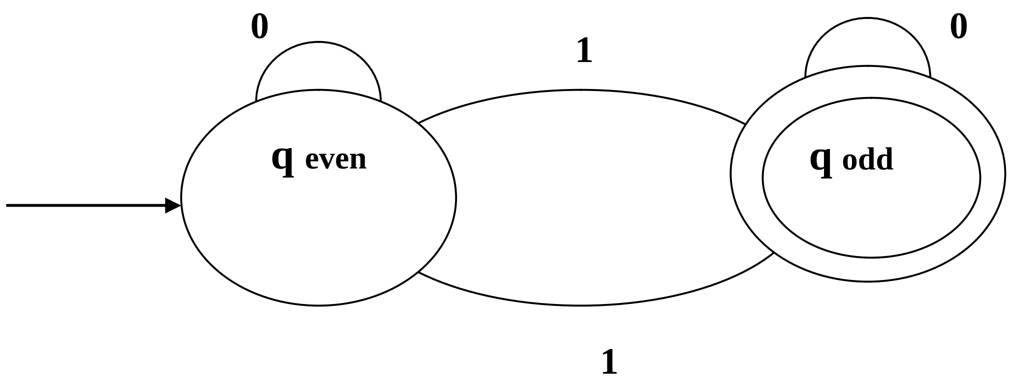
q odd **تمثل الحالة** التي يكون فيها عدد الحالات التي تمت قراءتها فردي

الخطوة التالية هي ضبط عملية الانتقال بين الحالتين بعد قراءة كل رمز في المتسلسلة. ونلاحظ هنا أنه **بعد قراءة الرمز 0 فإن عدد الآحاد** لن يتغير وهذا يعني أن الآلة سوف تبقى في نفس الحالة الموجودة فيها فعلا **وعند قراءة الرمز 1** فأن الآلة سوف تنتقل من الحالة الموجودة فيها إلى الحالة الأخرى.

بعد هذه المرحلة يجب تحديد الحالة الابتدائية التي سوف تبدأ الآلة عملها منها. **الرمز الأول الذي سوف تبدأ** الآلة بقراءته هو 0 وهو يعتبر عدد زوجي لذلك سنعتمد الحالة الابتدائية q even كما هو موضح أدناه:



وأخيرا يبقى اختيار حالة القبول Accept State وهي بلا شك ستكون الحالة q odd لأنها هي الحالة **المطلوبة**. وبذلك يصبح الشكل النهائي لمخطط هذه **الآلة على النحو التالي**:



مثال: سوف نقوم في هذا المثال بتصميم Finite Automaton وليكن E1 بحيث تكون هذه الآلة قادرة على إدراك لغة هي عبارة عن جميع المتسلسلات التي تحتوي على المتسلسلة الفرعية 001 **مثل المتسلسلات** : 0010, 1001, 001, 1110011 وهكذا حيث أن جميع المتسلسلات جزء من هذه اللغة. **في** حين المتسللات 11, 0000, 11110 **لا تعتبر** جزءاً من هذه اللغة.

يتلخص مبدأ عمل هذه الآلة في أنه عندما تبدأ الرموز بالدخول فإن الآلة سوف تتجاهل جميع الآحاد التي يتم إدخالها أي أنها لن تقوم بتذكرها. **أما إذا تم إدخال الرمز** 0 فإنها ستقوم بتذكر هذه الحالة على اعتبار أن هذا الرمز هو أول رمز من الرموز الثلاثة في المتسلسلة الفرعية المطلوبة. عند هذه النقطة إذا تم إدخال **الرمز** 1 فإن الآلة ستقوم بتجاهل هذا الرمز وتعود الآلة إلى الحالة الابتدائية أي في انتظار إدخال 0 **من** جديد. **أما إذا تم إدخال** 0 **ثاني** فإن الآلة سوف تتذكر هذه الحالة على اعتبار أنه تم استقبال رمزين من **الرموز الثلاثة المطلوبة**. **بعد ذلك إذا تم إدخال** 0 من جديد فإن الآلة سوف تبقى في نفس الحالة في انتظار وصول الرمز الأخير وهو 1. **وعند وصول الرمز** 1 وبغض النظر عن عدد الأصفار التي تم إدخالها قبله **فإن الآلة** سوف تتذكر هذه الحالة على أنها الحالة المطلوبة وهي متسلسلة فرعية مكونة من 001. **بعد ذلك ستبقى الآلة** في نفس الحالة مهما كانت الرموز التي يتم إدخالها.

بناءاً على ما سبق فإن هناك أربع حالات هي:

1. عدم مشاهدة أي رمز من رموز المتسلسلة الفرعية
2. مشاهدة الرمز 0
3. مشاهدة الرمزين 00

4. مشاهدة الرموز 001

وهذه الحالات جميعها يمكن تمثيل كل منها على النحو التالي:

q : الحالة الابتدائية (وهي حالة عدم مشاهدة أي رمز من رموز المتسلسلة المطلوبة).

q0 : حالة مشاهدة الرمز 0 .

q00 : حالة مشاهدة الرمزين 00 .

q001 : حالة مشاهدة الرموز 001 (حالة مشاهدة المتسلسلة الفرعية المطلوبة).

أما عملية الانتقال بين الحالات فسوف تكون على النحو التالي:

1. في الحالة الابتدائية q : **إذا تم قراءة الرمز** 1 فإن الانتقال سيتم إلى الحالة نفسها وإذا تم قراءة 0 فسيتم **الانتقال**

إلى الحالة q0 .

2. **في الحالة** q0 : **إذا تم قراءة** 1 يتم الانتقال إلى الحالة الابتدائية q **وإذا تم قراءة** 0 يتم الانتقال إلى الحالة q00 .
3. **في الحالة** q00: **بعد قراءة الرمز** 1 يتم الانتقال إلى q001 في حين قراءة 0 تبقي الآلة في الحالة نفسها.
4. **في الحالة** q001 :**قراءة الرمز** 0 **أو قراءة الرمز** 1 سوف تبقي الآلة في الحالة نفسها.

أما حالة القبول فهي الحالة q001 . **وبذلك** يكون مخطط الحالة لهذه الآلة على النحو التالي:

